Sobel Based Edge Detection

Final Report

Pradeep Lam
Tatparya Shankar
Sthitapragyan Parida
Kareem El Azhary

ECE337

Lab Section: Wednesday 11:30

Date: 5/5/2016

TA Name: Nick Pfister

Signatures
Pradeep Lam
Tatparya Shankar
Sthitapragyan Parida
Kareem El Azhary

1. Executive Summary

Our design utilizes an FPGA to quickly and accurately compute the edges of a given input image and output the results via vga to a display. The system will take the input image from the internal memory of the FPGA, grayscale it, and then, implement the Sobel edge detection algorithm to produce an output image containing only the edges from the input. Block RAM will be used to store the image and the read operation will allow us to read in the pixels of the image and perform our algorithm on them. We will use block RAM as opposed to distributed RAM because of the extra space, faster handling (since distributed RAM might cause extra wiring delays and increased synthesis time with larger images) and the read operation is synchronous with block RAM.

A market for this device is emerging in the field of computer vision, more specifically in applications to robotics and OCR. As the field of robotics advances it is necessary for faster techniques to detect edges as to allow robots to emulate real time human vision. In regards to OCR, google is in the process of refining a real time translation mechanism which allows for in place translation of text from one language to another. A key subprocess in OCR is the detection of edges. In both these cases, the speed of computation is key.

In regards to product differentiation, an ASIC based design allows for the efficient computation of edges, much faster than can be programmed using software.

In regards to materials, we will require a FPGA. The rest of the proposal covers design specifications and design pinout and design architecture.

2 Design Specifications

2.1 System Usage

2.1.1 System Usage Diagram

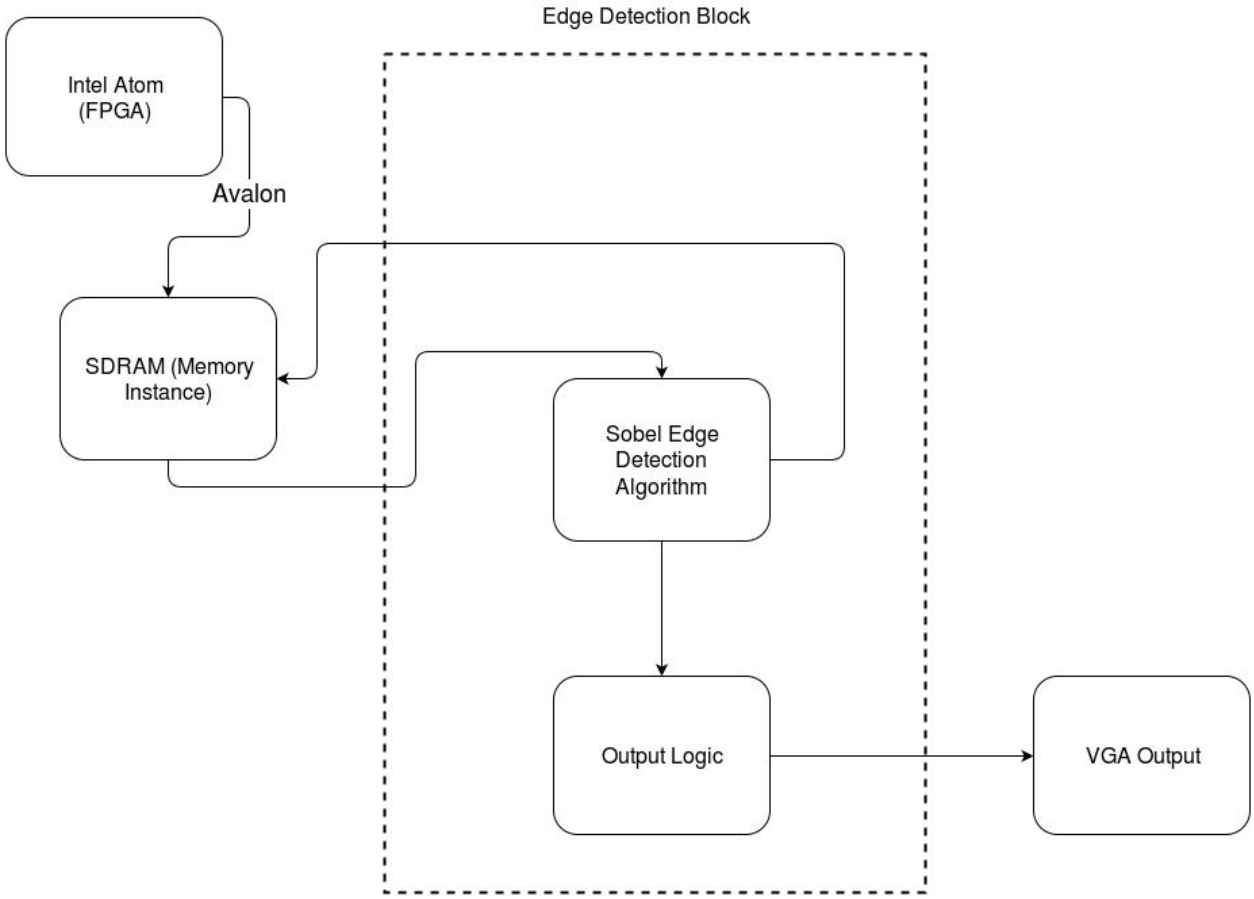


Fig. System Usage Diagram

2.1.2 Implemented Standards and Algorithms Overview

As mentioned before, the SDRAM on the FPGA was used to store the image as opposed to distributed RAM. The PCIe interface was move the image from the usb port to the chipset which will then go to the atom. From there the avalon bus will provide the image to the rest of the blocks from the SDRAM.

Image Standards:

Bitmap: 640 * 480 Bitmap *(.bmp)* images were used as inputs to the edge detection filter. The exact specifications of the image type follows:

Header Size : 40 bytes

Width: 640 Pixels

Height: 480 Pixels

Planes: 1

Bits: 32 bits per pixel

Compression: BI_RGB (None)

Colour Table: RGBA32

Image Size: 1228800

X Resolution: 2835 Pixels/Meter

Y Resolution: 2835 Pixels/Meter

Image Processing Algorithms:

Grayscale Conversion: The original Colour image will be converted into a grayscale image using the luminosity method. The details are as follows:

$$\text{Pixel Value} = (0.21\ \text{Pixel\_R} + 0.72\ \text{Pixel\_G} + 0.07\ \text{Pixel\_B})$$

Sobel Edge Detector: Separate horizontal and vertical gradient detection kernels will be used in the algorithm. The specific kernels used are as follows.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Fig. Sobel Gradient Matrices

After applying the kernels, to avoid taking square roots and invoke trigonometric lookup tables, the absolute gradient magnitude will be approximated using the formula:

$|G| = |G_x| + |G_y|$

Output Interfaces:

VGA: We will use the VGA interface on the DE2i-150 to output the line gradient image to a vga cable and onto a screen. The specific graphics modes used are as follows:

Pixels: 640 * 480

Color: Monochrome

Refresh Rate: 60 Hz

2.2 Design Pinout

Common Pins

Signal Name	Type In/Out/ Bidir.	Number of Bits	Description
Clk	Input	1	Clocks the Circuit
Rst	Input	1	Asynchronously Reset the Circuit
Gnd	Input	1	Grounds the Circuit
Pwr	Input	1	Powers the Circuit
VGA	Output	15	Output Image to Display

Interface Pins

Signal Name	Type In/Out/ Bidir.	Number of Bits	Description
startSignal	Input	32	Starts the Processing of Image
inputData	Input	32	Input Pixel Data from FPGA in Form 0RGB (Hex)
rwError	Input	32	Input from FPGA Read/Write Error (Error => 0001 hex)
readDone	Input	32	Input Read Done from FPGA (When Address Has Been Processed & inputData Ready for Program)
writeDone	Input	32	Input Write Done from FPGA (After Address Has Been Processed & outputData Has Been Read by FPGA)
readAddress	Output	32	Address offset for FPGA to Read

writeAddress	Output	32	Address offset (count) for FPGA to Write
outputData	Output	32	Data for FPGA to Write in Form 000S hex (S=sobel pixel)
readEnable	Output	32	Flag Telling FPGA to Read Address & Send Back Pixel
writeEnable	Output	32	Flag Telling FPGA to Read Address & outputData
outputError	Output	32	Flag Telling FPGA That Internal Error Has Occurred Somewhere

2.3 Operational Characteristics

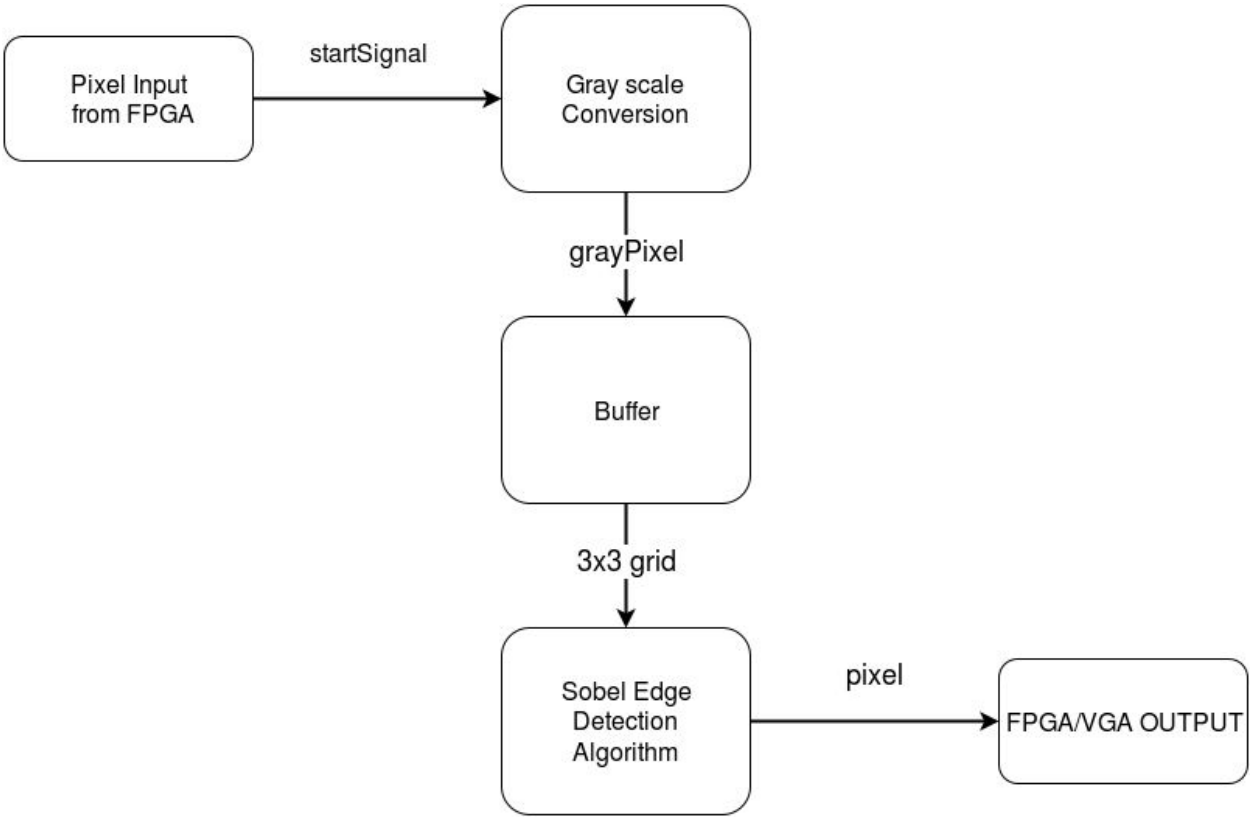
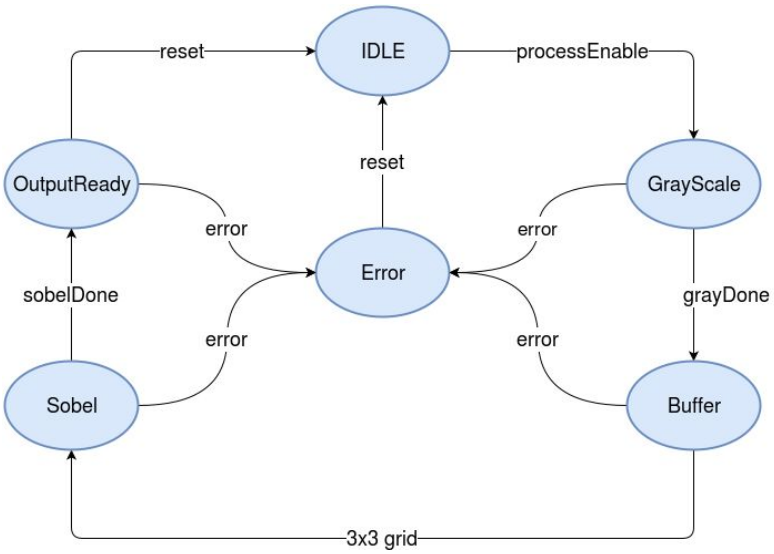


Fig. Macro-Level Diagram

Flow Diagram:



IDLE	OutputReady
resetSystem	vgaEnable
error	error
GrayScale	**Sobel**
grayDone	sobelDone
error	error
Buffer	
3x3 grid	
error	

Fig. Macro-Level Timing Diagram

Explanations:

FPGA:

We have 3 distinct parts to our system. The C application program, the avalon interface written in verilog and the image detection algorithm that uses the avalon interface to interact with image data stored in SDRAM. The C program resides in the Intel Atom memory and handles the overall execution of the program.

Initially the C program loads a 24 bit bitmap colour image into memory using the C program. The pixel values are then written into the FPGA SDRAM using the slave interface running on the Avalon bus. Initially we tried to use the DMA_Write block function but failed to successfully write pixel values to memory. Then we realised, that pixel values written were not accurate and we resorted to writing a single pixel value at a time to the CSR registers and then copying the pixel to the appropriate location is SDRAM memory by using the custom interface. After the image loading is complete, the C program asserts a start process byte(0x53) to a predetermined csr register to signal the avalon verilog interface to move forward with the actual execution of the image detection algorithm. The C program now waits for the stop byte (0x12) from the custom interface to stop execution.

Once the verilog interface(custom_master_slave) receives the start byte, it moves to a STARTPROCESS state. At this point it checks for the flag controlled by the on board switch 1 to determine whether the edge detected version or the actual image should be displayed on the VGA display. If the switch is asserted, the custom master slave simply copies the image stored from 0x09000000 - 0x0912C000 in FPGA SDRAM to the VGA display memory (0x08000000 - 0x0812C000) and moves to assert the WRITESTOPBYTE state discussed below. Otherwise, the module asserts the start conversion flag to signal the custom edge detection module that the image data is valid and ready to be processed.

The custom edge detection block written in verilog interacts with the SDRAM using the READ and WRITE interfaces. The execution of the block are described in Section () (WHAT SECTION IS THIS?) During execution, to process each pixel, the edge detection block retrieves the values of 9 pixels from SDRAM (starting from 0x09000000) and processes them to produce the final pixel value of the edge detected image. This pixel is written down serially (as we

perform a raster scan) into the memory location in SDRAM mapped to the VGA display (0x08000000 - 0x0812C000). After all (640 - 1) * (480 - 1) pixels are processed, the custom master slave program moves to WRITESTOPBYTE state to assert a stop signal which signals the C program to terminate execution. At this point the line detected version of the image is displayed on the FPGA VGA display.

Algorithm:

Controller: The controller remains in idle state until a start signal is asserted. Once it is asserted, the controller moves on to a state that generates an address offset for the next pixel to be loaded. When the address is generated, the readEnable signal is asserted to allow the FPGA to read the address offset. The controller waits until a readDone signal is asserted and then waits until the pixel is grayed. Once it is grayed, there is a state that controls storing the grayed pixel in a buffer. This state also waits until the buffer has enough pixels for the sobel algorithm. Once that is done, the writeEnable signal is asserted and the controller waits until the writeDone is asserted and then the process restarts by generating another address. There is also an error state in case anything goes wrong with reading/writing.

Generate Address Offset: There is a controller and counter dedicated to calculating the address offset from the base address to tell the FPGA which pixel to send. This program also determines whether 3 or 9 pixels are needed. It does that by counting where we are in a certain row/column.

Grayscale Conversion: An RGB pixel is input to this module and a gray pixel value is calculated as described above and sent out to be added to the buffer.

Flex Counter: A flex counter is used to keep track of the pixels being stored in the buffer. Depending on how many pixels we need, it counts up to either 3 or 9 and that is used as an "index" of where the pixel will be stored in the buffer (matrix).

Buffer: The buffer works by concatenating the bytes that come from the grayscale conversion module. It has an input that tells it whether it should add 3 pixels or 9.

Sobel Algorithm: The sobel algorithm takes in a 9x9 matrix and uses the formulas shown above to calculate the sobel value and outputs it.

Capture Output: This module waits for a signal that tells it when the sobel value is ready and it holds it until the data is read. It also asserts a signal that is used to assert the writeEnable signal.

2.4 Requirements

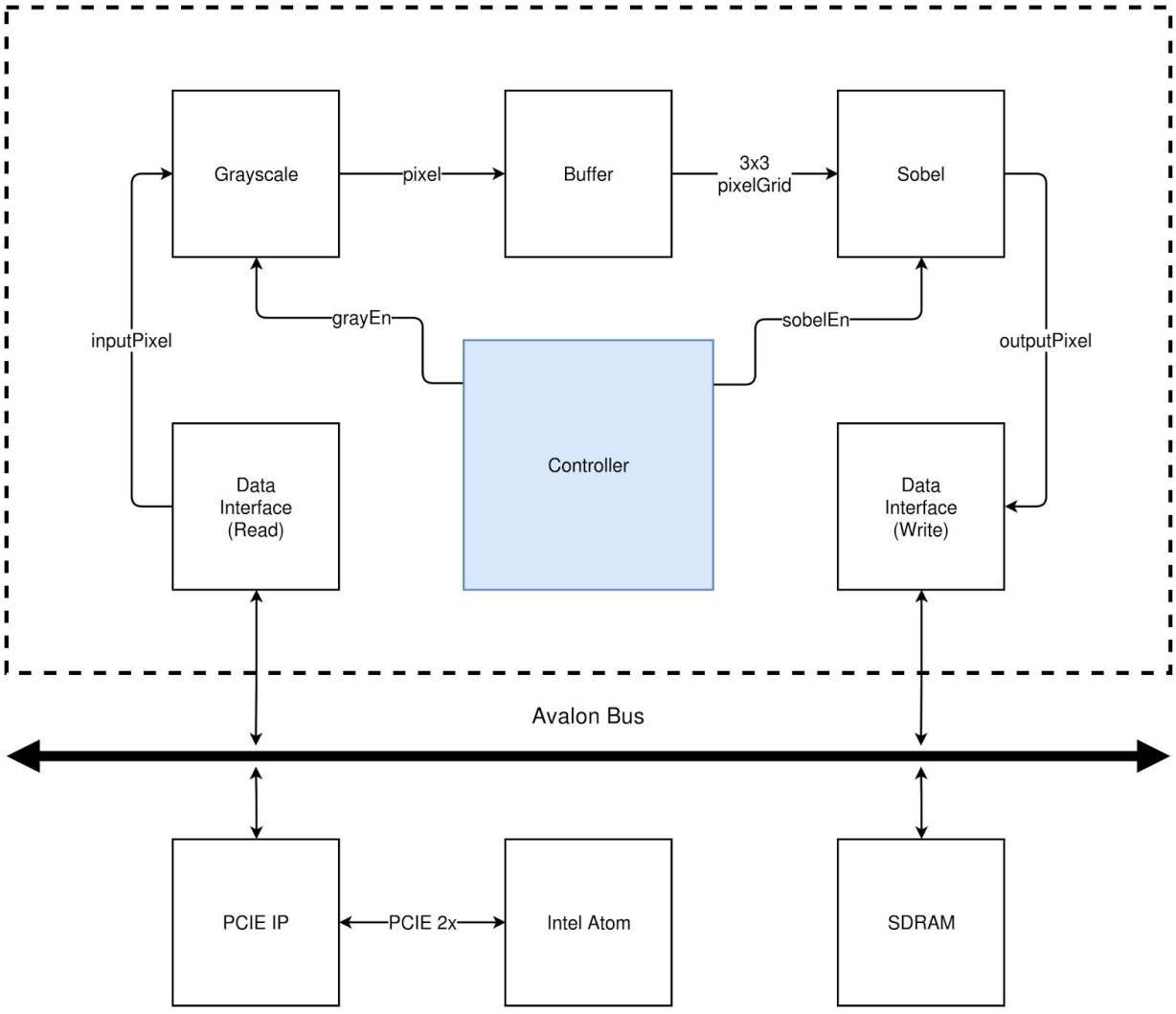
The special requirements for this design are geared towards the needs of our application. The purpose of implementing edge detection on an ASIC rather than in pure software is to decrease the runtime. Computational latency is,therefore, our highest priority in comparison to other qualities specified. In order to ensure that the algorithm runs efficiently, we must pay attention to the organization of design and how it affects critical path. Design specifications geared towards efficiency include optimizing pre-processing logic and accelerated matrix handling.

For instance, a buffer based raster scan system will be implemented as to reuse as many pixels as possible in calculating edges in order to minimize the number of pixels for which the design will need to pull from memory. Also, the image file will be stored in block RAM as opposed to distributed RAM as to allow for faster synthesis and to reduce potential extra wiring delays. Also, the read operation is synchronous with block RAM so the data signal will not have to be synchronized when reading. On the other hand, Area is a very low priority. We are using an fpga, and this minimizes the chances of defects caused by manufacturing. Power usage isn't an issue for such low bandwidth of computation. Our aim is to achieve 50 MHz with a pin count of 13.

3 Design Implementation

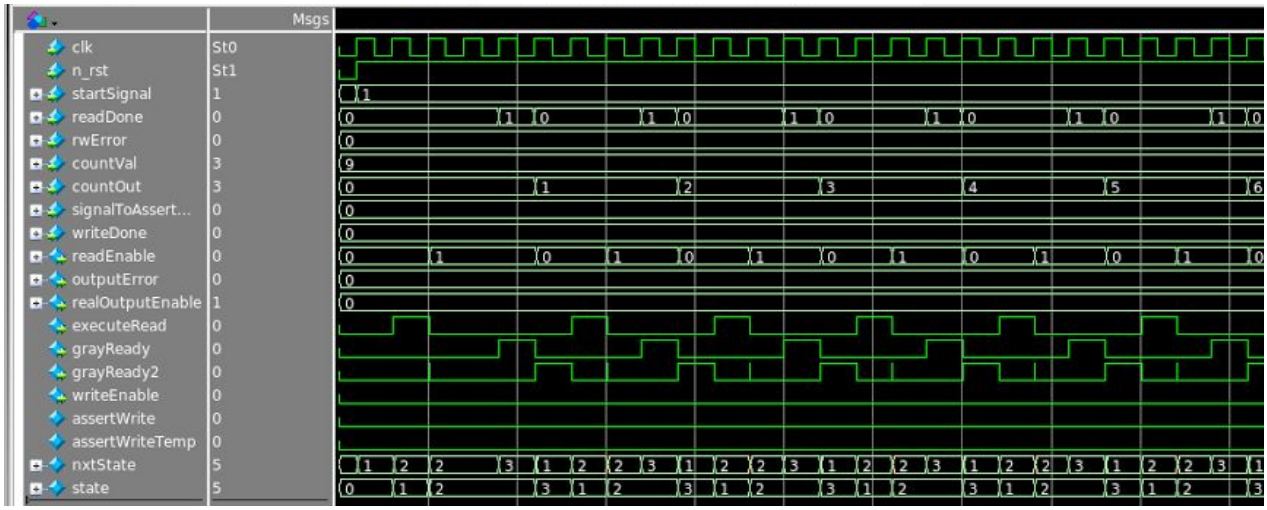
3.1 Design Architecture

Architectural Block Diagram :



Timing Diagrams

** *For the following timing diagrams, please refer to the operational characteristics section for a description of how each module works (explanatory text).*



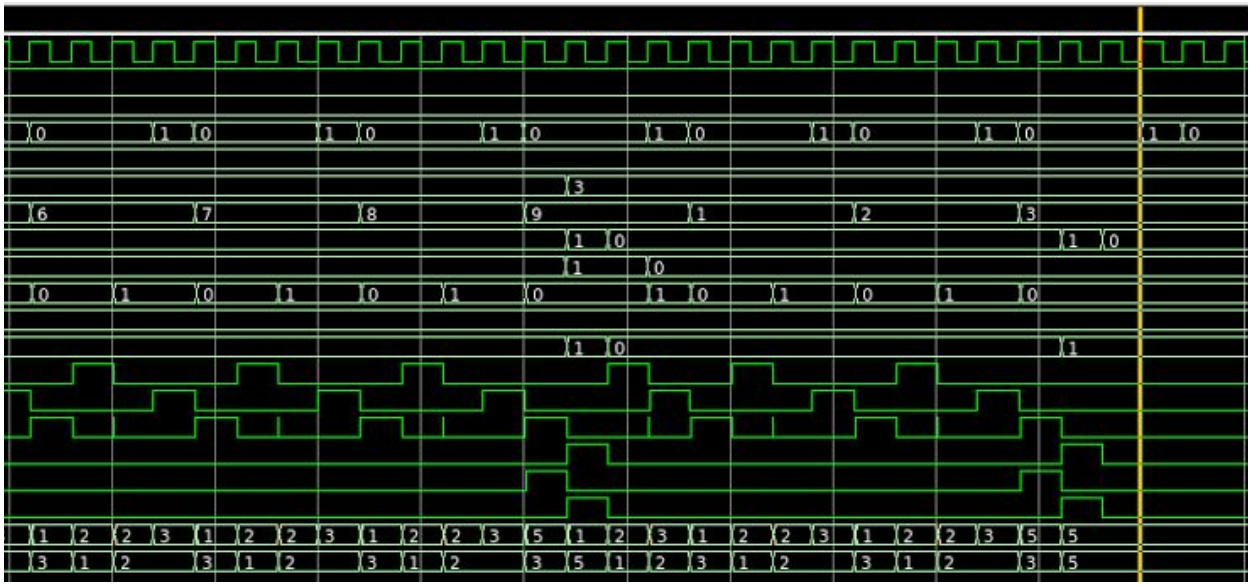


Fig. Above is the timing diagram of the controller

Fig 1.

Fig 2.

Fig 3.

Fig 4.

Fig 5.

Fig 6.

Fig 1. : Timing diagram of the capture output module.
Fig 2. : Timing diagram of the sobel calculation module.
Fig 3. : Timing diagram of the buffer module.
Fig 4. : Timing diagram of the address offset generator module.
Fig 5. : Timing diagram of the grayscale module.
Fig 6. : Timing diagram of the overall algorithm module. It shows the first two sobel values calculated. This shows that we can read 9 pixels first and then only add 3 pixels use 6 old ones to calculate the next sobel value.

3.2 Design Modules and Components

FPGA Interfacing RTL Diagram

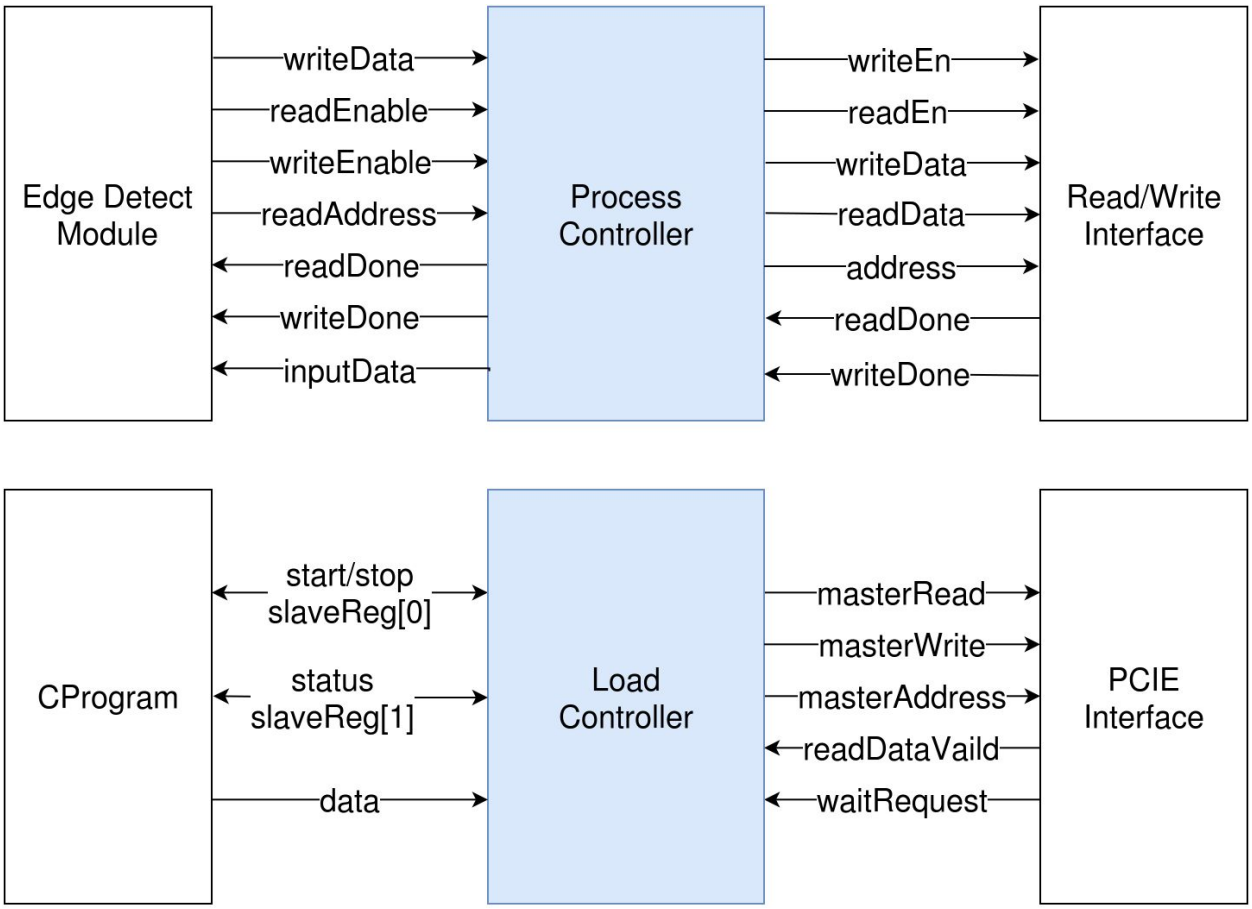


Fig. FPGA Interface RTL Diagrams

Image Loading

Slave Register Interface

Start / Stop	Status	Data
Start	CopyPixel	PixelData
Stop	LastPixel	
	NextPixel	

Fig. Slave Register Interface Table

The slave interface above was used to use the Avalon slave interface as a means to transfer the data between the atom memory and the SDRAM. The first slave register was used as the start or stop bit to tell the image filtering process to start or stop the process. The status register on the slave was used to communicate between the loading controller and the C program to assert statuses based on the next required action. The data register on the slave register was simply used to transfer the pixel data into the SDRAM memory during the CopyPixel cycle. Using the above interface, an image was successfully copied from the atom memory to the SDRAM on the FPGA. The Load controller then asserted the process enable flag to indicate the start of the sobel filter algorithm to process the image in the SDRAM.

SDRAM Memory Interface

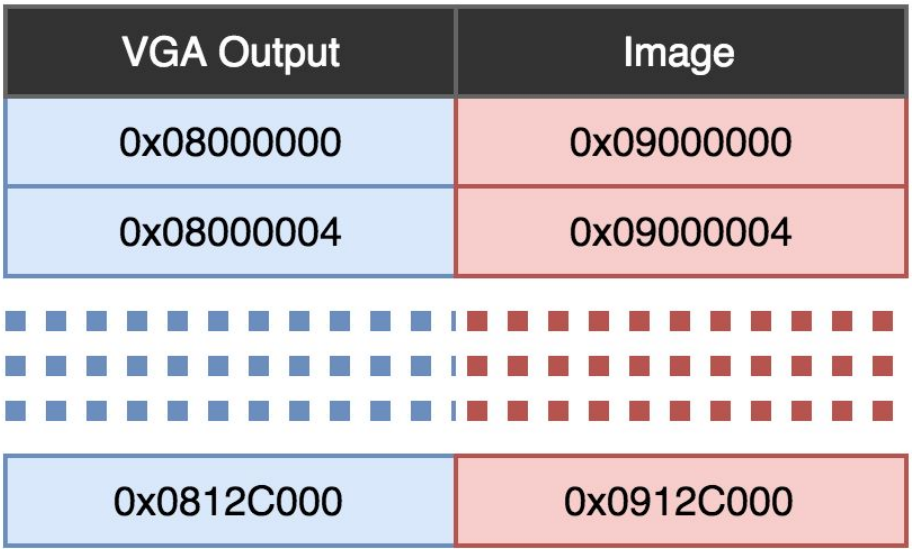


Fig. SDRAM Memory Management Table

The above memory interface was used for the SDRAM. The original image was uploaded into the SDRAM at locations 0x09000000 through 0x0912C000. The process controller then sequentially provided pixels from the SDRAM memory to the image filter. The processed sobel image was stored from locations 0.08000000 through 0x0812C000 which was also used as the VGA Output to be able to output the final image with the detected edges on to the VGA screen connected to the FPGA.

Functional Block Diagram:

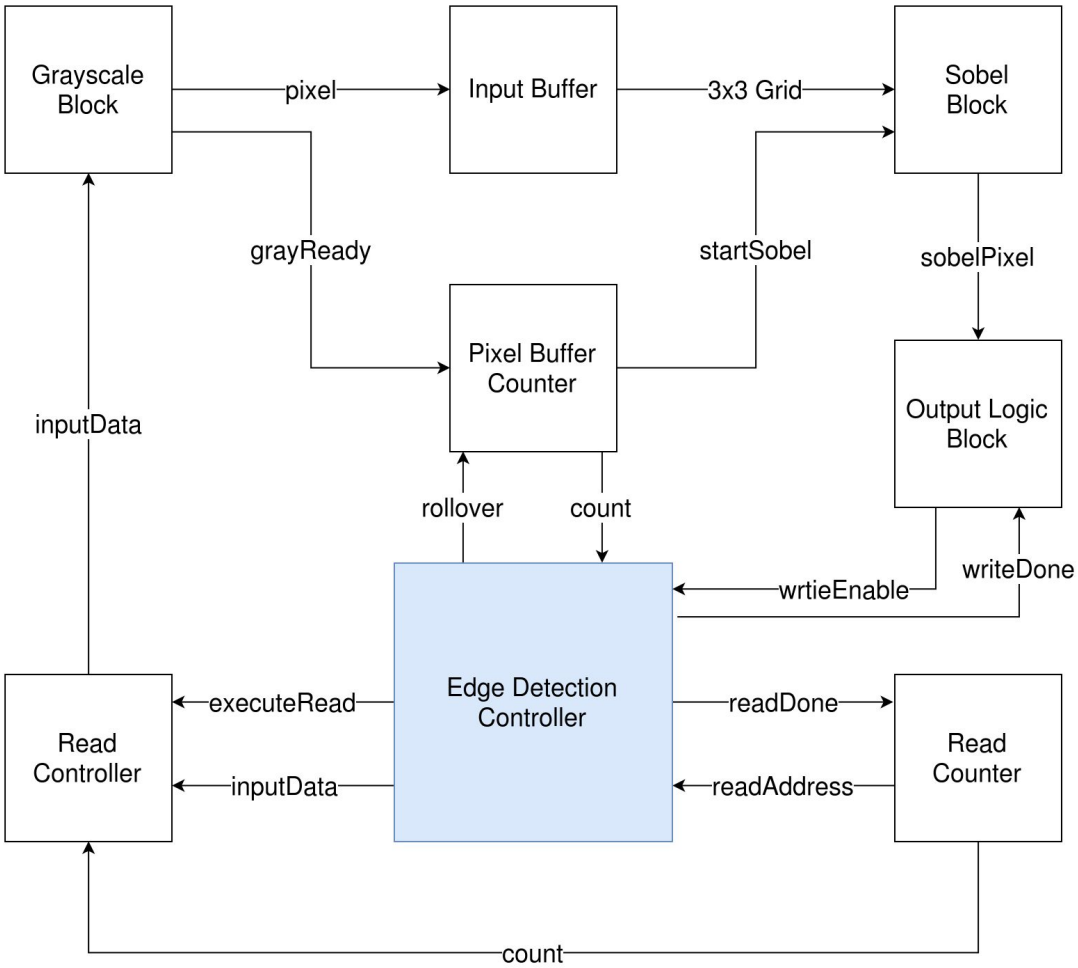


Fig. Edge Detection Module Functional Block Diagram

** ***All functional blocks are described in the operational characteristics section. Timing diagrams can be found in design architecture*** **

Area Analysis

Name of Block	Category	Gate/FF Count	Err:501	Comments
read address	Combinational	280	210,000	read address
write address	Combinational	160	120,000	
step counter 1 nxt-cnt	Combinational	6	4,500	for rgb rollover is 3
step counter 1 cnt-reg	Reg. w/ Reset	2	4,800	
step counter 1 out-logic	Combinational	2	1,500	
step counter 2 cnt-reg	Reg. w/ Reset	5	12,000	
step counter 2 out-logic	Combinational	5	3,750	
step counter 3 nxt-cnt	Combinational	12	9,000	for sobel rollover is 18
step counter 3 cnt-reg	Reg. w/ Reset	5	12,000	
step counter 3 out-logic	Combinational	5	3,750	
Shift 1	Reg. w/ Reset	24	57,600	for rgb 3 bytes
Shift 3	Reg. w/ Reset	72	172,800	for sobel 9 bytes
process block 1	Combinational	456	342,000	for rgb
process block 3	Combinational	2656	1,992,000	for sobel
Controller 1 nxt-state	Combinational	8	6,000	for rgb 8 states
Controller 1 state	Reg. w/ Reset	3	7,200	

Controller 1 out-logic	Combinational	3	2,250	
Controller 2 state	Reg. w/ Reset	4	9,600	
Controller 2 out-logic	Combinational	4	3,000	
Controller 3 nxt-state	Combinational	10	7,500	for sobel 10 bytes
Controller 3 state	Reg. w/ Reset	4	9,600	
Controller 3 out-logic	Combinational	4	3,000	
			N/A	
			N/A	
			N/A	
			N/A	
			N/A	
Total Core Area			6,280,350	

Chip Area Calculations (units in um or um2)				
Number of I/O Pads:	12			
I/O Pad Dimensions:	90	by	300	
I/O Based Padframe Dimensions:	870	by	870	
Core Dimensions	2,506	by	2,506	
Core Based Padframe Dimensions:	3,406	by	3,406	

Final Padframe Dimensions:	**3,406**	**by**	**3,406**	[redacted]
Final Chip Area:	**11,601,263**			[redacted]

Fitter Summary

Fitter Status	Successful - Thu May 5 01:48:03 2016
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Full Version
Revision Name	master_example
Top-level Entity Name	master_example
Family	Cyclone IV GX
Device	EP4CGX150DF31C7
Timing Models	Final
Total logic elements	11,646 / 149,760 (8 %)
Total combinational functions	9,161 / 149,760 (6 %)
Dedicated logic registers	7,757 / 149,760 (5 %)
Total registers	7875
Total pins	200 / 508 (39 %)
Total virtual pins	0
Total memory bits	98,276 / 6,635,520 (1 %)
Embedded Multiplier 9-bit elements	0 / 720 (0 %)
Total GXB Receiver Channel PCS	1 / 8 (13 %)
Total GXB Receiver Channel PMA	1 / 8 (13 %)
Total GXB Transmitter Channel PCS	1 / 8 (13 %)
Total GXB Transmitter Channel PMA	1 / 8 (13 %)
Total PLLs	2 / 8 (25 %)

# of Lookup Tables	9161
# of Flip Flops	7757
Max Logic Elements	16918
Min Logic Elements	9161
Area Estimates	2114750 um^2 [Within Constraints]
Max Area According to Design Constraint	11601263 um^2

3.3 Design Timing Analysis

Starting Component	Propagation Delay (ns)	Combinational Logic	Propagation Delay (ns)	Ending Component	Setup Time or Propagation Delay (ns)	Total Path Delay (ns)	Target Clock Period (ns)	
control unit	40	read,shift,process,count	660	write address/ control unit	20	720	20	for rgb
control unit	40	read,shift,sobelx,sobely, magnitude,count	820	write address/ control unit	20	880	20	for sobel

Since the RTL's almost follow the same format, the main difference in timing is the processing block. In the table above, the critical paths for the two algorithms are shown.

Fast 1200mV 0C Model Setup Summary

	Clock	Slack	End Point TNS
1	amm_master_inst|pcie_ip|pcie_interna...i.cycloneiv_hssi_pcie_hip|coreclkout	3.532	0.000
2	clock_50_1	13.109	0.000
3	amm_master_inst|altpll_qsys|sd1|pll7|clk[1]	14.416	0.000
4	n/a	17.001	0.000
5	amm_master_inst|altpll_qsys|sd1|pll7|clk[2]	37.367	0.000

Fast 1200mV 0C Model Hold Summary

	Clock	Slack	End Point TNS
1	amm_master_inst|pcie_ip|pcie_interna...i.cycloneiv_hssi_pcie_hip|coreclkout	0.028	0.000
2	amm_master_inst|altpll_qsys|sd1|pll7|clk[1]	0.079	0.000
3	clock_50_1	0.167	0.000
4	amm_master_inst|altpll_qsys|sd1|pll7|clk[2]	0.172	0.000
5	n/a	2.672	0.000

4 Success Criteria

Fixed Criteria

1. Test benches exist for all top level components and the entire design. The test benches for the entire design can be demonstrated or documented to cover all of the functional requirements given in the design specific success criteria.
2. Entire design synthesizes completely, without any inferred latches, timing arcs,and, sensitivity list warnings.
3. Source and mapped version of the complete design behave the same for all test cases. The mapped version simulates without timing errors except at time zero.
4. A complete IC layout is produced that passes all geometry and connectivity checks.
5. The entire design complies with targets for area, pin count, throughput (if applicable), and clock rate. The final targets for these parameters will be determined by course staff based on your design review. Failure to reach any of the targets will result a score of 1 out of 2 provided that you are within 50% on area, 10% on pin count, and 25% on throughput. Doing worse in any category will result in a score of 0.
    1. Area: 4mm x 4mm
    2. Pin Count: 13 Pins (10 Pin Edge Detect, Power, Ground, Clock)
    3. Clock Period: 50 MHz

Status of Fixed Criteria

1. Complete - There exist tb for all top level components and the entire design. The test benches for the entire design can be demonstrated to cover all of the functional requirements given in the design specific success criteria.
2. Complete - No latches, timing arcs, and sensitivity list warnings exits.
3. Complete - Source and mapped version of the design behave the same for all test cases. The mapped version simulates without timing errors.
4. Complete - Design uses FPGA.
5. The entire design compiles with targets for area, pin count, throughput, and clock rate.

Proof for Fixed Criteria

Refer to verification section of report. All criteria are demonstrated through FPGA run.

Design Specific Criteria

1. Demonstrate by simulation of a verilog test bench or fpga that the grayscale conversion successfully converts RGB images to black and white images. (2 Point)

2. Demonstrate by simulation of verilog test bench or fpga that the multiplication and addition modules work accurately. (1 Point)
3. Demonstration by simulation of a verilog test bench or fpga that Sobel Edge Detection produces visible and convincing edge detection. (2 Point)
4. Demonstrate by display on a computer screen that the VGA output display of the image works successfully and image data can be written onto SD RAM and then displayed. (2 Point)
5. Demonstrate by comparison with a python script that the edge detection is faster than pure software implementation. (1 Point)

Status of Design Specific Criteria

1. Complete - Demonstrates that the grayscale conversion successfully converts RGB pixels to black and white pixels..
2. Complete - Demonstrates that multiplication and addition modules work accurately.
3. Complete - Demonstrates that Sobel Edge Detection produces visible and convincing edge detection.
4. Complete - Demonstrates by display on computer screen that the VGA output display of the image works successfully and image data can be written onto SD RAM and then displayed.
5. Complete - Demonstrated by comparison with a python script that the edge detection is faster than pure software implementation.

5 Design Verification

5.1 Design Verification Overview

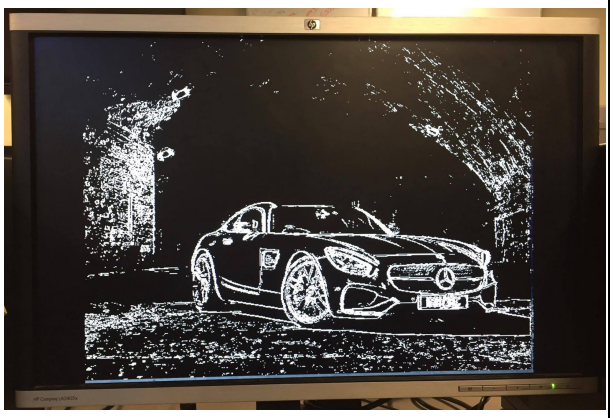
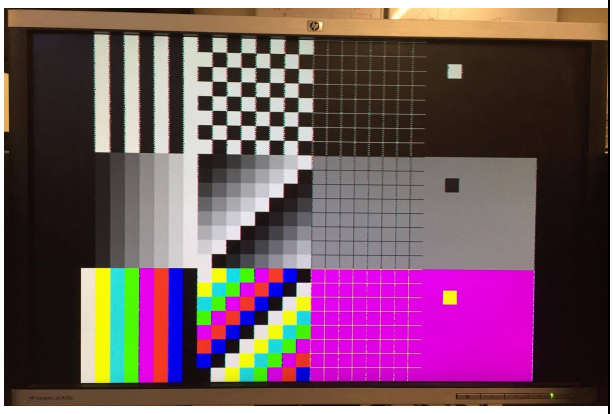
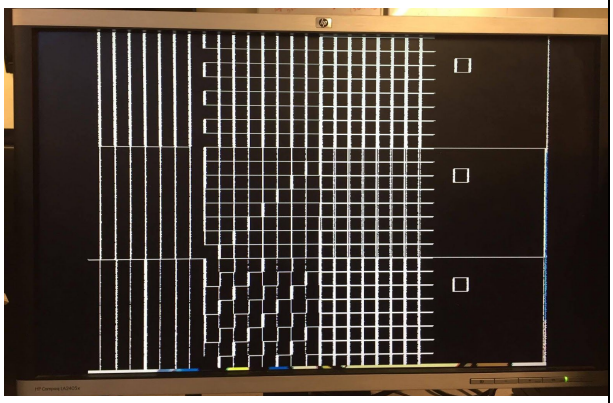
1. Test whether image can be loaded onto memory of FPGA and run through the design and that the resulting gray edge-detected image can be displayed on the monitor.
    a. DSSC 1,2,3,4
    b. Run Several Images Through Process - See If Edge Works
    c. Involves All Modules Part of Design
2. Compare the estimated run time of the program with the runtime of a python script.
    a. Run Several Images Through Process - Compare Run Times to Calculated Run Time for Design
    b. Involves No Modules

5.2 Test Scenario Breakouts

1. Load images into the FGPA.
2. Run them through design.
3. Observe the output and see if edges are detected correctly.
4. Run same images through a python program and observe calculation time.

5.2 Test Scenario Breakouts

FPGA Sobel Edge Detection Results

	
Test Image 1	Test Image 1 Result
	
Test Image 2	Test Image 2 Result
	

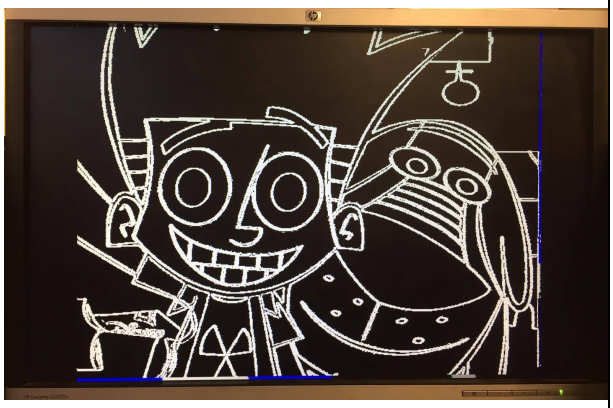
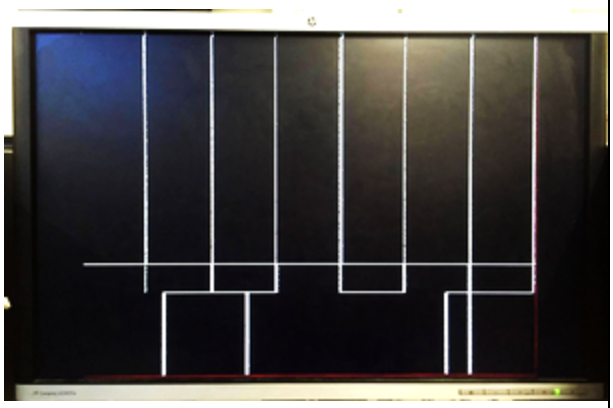
Test Image 3	Test Image 3 Result
	
Test Image 4	Test Image 4 Result
	
Test Image 5	Test Image 5 Result

Table. FPGA Sobel VGA Output Results

Python Sobel Edge Detection Results ***(For result comparison)***

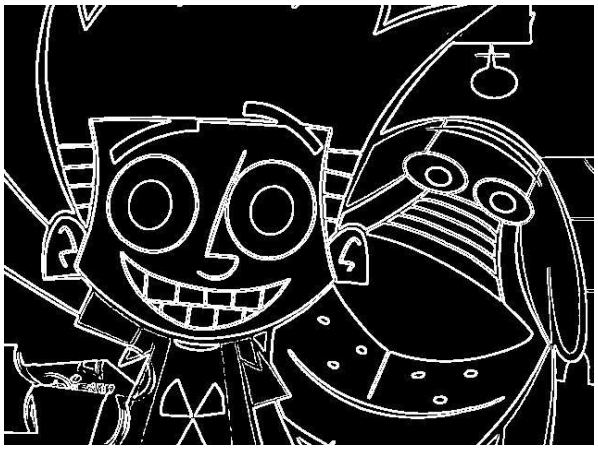
	
Test Image 1	Test Image 1 Result
	
Test Image 2	Test Image 2 Result

Table. Python Sobel Edge Detection Results

Run-time Rundown

```
ee364c14@ecegrid-thin8 Lab11 1009 vi sobel.py
ee364c14@ecegrid-thin8 Lab11 1010 python3 sobel.py
Total running time: 2.8610475063323975 seconds
ee364c14@ecegrid-thin8 Lab11 1011 vi sobel.py
ee364c14@ecegrid-thin8 Lab11 1012 python3 sobel.py
Total running time: 2.7883410453796387 seconds
```

Fig. Python Script runtime

Maximum # of clock cycles processing a single pixel value = 16

Max time to process all pixels = 16 x 1/50Mhz x 640 x 480 = 0.098304 sec

Average Time for Python program to execute same algorithm on Linux machine = 3.1 sec

Percentage Improvement = 271%

Significant Design Decision

Removal of gaussian module

During the design phase our project, we has two serial subprocesses to obtain the line detected version. First we planned to pass the image through a Gaussian Blur and then further move on to the Sobel Edge Detection Algorithm. However, at the end of the day we decided on not using the Gaussian Blur before passing the grayscale image to the Sobel Edge Detection Block. During initial testing using a python script version of our algorithm, we noticed that the edges that popped up in the final image were sensitive to the Sigma value (Standard Deviation) used in the Gaussian Filter. A single value did not produce the best results for every image and we had to manually tweak the Sigma value to obtain ideal results and clear edges. After more research, we realised that in case of no noise images, the Gaussian Blur actually blurred out the edges (Adjacent Figure) and reduced their intensity and therefore the Sobel edge detection was unable to pick up on the reduced gradient. We verified this face by removing the Gaussian blur from the algorithm chain entirely and performed the Sobel edge detection algorithm immediately on the grayscale version of the image. The results we found were consistent across different images. Since we did not have a control for the Sigma value in our verilog module and we didn not want to over complicate our design we decided to drop the Gaussian Module and directly move on to the Sobel Edge Detection. This limits us with respect to images which have a lot of noise, but we decided to save this improvement for a later phase of the project.

6 Actual Division of Tasks

Pradeep Lam - Design of algorithm
Kareem El Azhary - Design of algorithm
Worked together to write the modules to perform the sobel edge detection. Pradeep focused on address calculations and Kareem focused on data flow and calculations.
Tatparya Shankar - FPGA
Sthitapragyan Parida - FPGA
Worked together to configure FPGA to load image into SDRAM, communicate with verilog code, write calculated pixels, and display image through VGA.

7 Appendix

File Name	Purpose
projectFiles/edgeDetection2.sv	Overall wrapper file
projectFiles/captureOutput2.sv	Hold output till writing
projectFiles/concat2.sv	Concatenates pixels
projectFiles/controller2.sv	Overall controller
projectFiles/flex_counter.sv	Index for concat
projectFiles/flex_counter2.sv	Used for address offset
projectFiles/readController.sv	Generates address offset
projectFiles/readCounters.sv	Counts addresses
projectFiles/sobel.sv	Calculates sobel value
projectFiles/tb_concat2.sv	Test Bench
projectFiles/tb_edgeDetection2.sv	Test Bench
projectFiles/tb_sobel.sv	Test Bench
projectFiles/tb_rgbToGray.sv	Test Bench
projectFiles/tb_testRead.sv	Test Bench
software/PythonTest/sobel.py	Python test file

***** For a detailed listing, please see readme *****

** Address for access to source files **

/home/ecegrid/a/mg75/ece337/ECE337_Project